United International University (UIU)

Dept. of Computer Science & Engineering (CSE)

Final Exam :: Spring 2022

Course Code: CSE 1115 Course Title: Object Oriented Programming

Total Marks: **40** Time: **2 hours**

READ THIS CAREFULLY: Any examinee found adopting unfair means will be expelled from the trimester / program as per UIU disciplinary rules

Answer all the five questions from [Question 1 to 5]. There are two questions named Question 5. You need to answer one of them.

Question 1 [4 + 4]

A. Suggest changes to the provided three classes for the following tasks:

i. Make the variable "m" both accessible from "***method2***" of class A and read-only once initialized

ii. If any class inherits class "B", prevent the "method1" from being overridden. But it should be allowed to override if any class inherits from class "A" or "C"

iii. Make sure class "C" cannot be inherited

N.B. *Just write only the modified lines*

```
class A{
    int m = 10;
    void method1(int t){
        System.out.println(t);
    }
    static void method2(){
        // access m here
    }
}
```

```
class B extends A{
    void method1(int t){
        System.out.println(t);
    }
}
class C extends A{
    void method1(int t){
        System.out.println(t);
    }
}
```

B. Create and assign an object of Vehicle using **anonymous inner class** for both of **Line 6** and **7** to produce output "*Uses LPG*" from line number 9 and "*Uses Jet Fuel*" from Line number 10:

```
1 interface Vehicle {
2   void fuelType();
3 }
4 public class Spring {
5   public static void main(String[] args) {
6         Vehicle cng; // Write suitable anonymous class
7         Vehicle airplane; // Write suitable anonymous class
8
9         cng.fuelType(); // should print "Uses LPG"
10        airplane.fuelType(); // should print "Uses Jet Fuel"
11  }
12 }
```

Question 2 [8]

Suppose you have a file "***id.txt***" that contains the ids of multiple UIU students. Write a java code to write the odd ids in the id.txt file to another file called "***odd.txt***" and the even ids in the id.txt file to another file called "***even.txt***". Check the following example for clarification:

id.txt	odd.txt	even.txt
011001212 011002213 011003214 011004215 011005216	011002213 011004215	011001212 011003214 011005216

Question 3 [8]

Consider the following class "CreditCard". The **constructor** and the **withdraw method** should throw a **user-defined exception** named "InvalidTxnException" with proper messages for the following cases:

- If the amount value passed to the constructor/method is negative, they throw the InvalidTxnException with the following message:
  "***-5000*** *is not a valid amount for the requested transaction*"
  Here, -5000 is the value passed as the amount from main at ***Line No: 18***
- If the withdrawal of the amount passed to the withdraw method crosses the max credit limit, throw the InvalidTxnException with the following message:
  "***4000*** *cannot be withdrawn with current credit of* ***7000*** *for your limit of* ***10000***"
  Here, 4000 is the value passed as the amount from main at ***Line No: 25***

Now write the user-defined exception: **InvalidTxnException** class with a proper super call to set messages. And **rewrite only** the **constructor** and the **withdraw method** of CreditCard class to throw exceptions based on proper conditions. Also, **handle** the InvalidTxnException from main for ***Line No: 18*** & ***25*** using the **try-catch** keywords.

```
1 class CreditCard{
2     private double credit_limit;
3     private double credit_current;
4     public CreditCard(double limit){
5         // check and throw InvalidTxnException
6         credit_limit = limit;
7         credit_current = 0;
8     }
9
10    public void withdraw(double amount){
11        // check and throw InvalidTxnException
12        credit_current += amount;
13    }
14 }
15 public class Main {
16    public static void main(String args[]) {
17        // handle the proper exception here with try-catch
18        CreditCard c1 = new CreditCard(-5000);
19
20        CreditCard c2 = new CreditCard(10000);
21
22        c2.withdraw(7000);
23
24        // handle the proper exception here with try-catch
25        c2.withdraw(4000);
26    }
27 }
```

Question 4 [4 + 4]

For the following GUI question, **write a single code** to answer both a & b.

a) Complete the following code to create a GUI as shown in image 1. Assume all necessary classes are imported

b) Provide button click handling code so that when button 1 (top-left) is pressed, the buttons show texts like image 1. When button 4 (bottom-right) is pressed, the buttons show texts like image 2

```
public class GUI {
    public static void main(String[] args) {
        JFrame frame = new JFrame("My App");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(300, 300);
        // Your code here
        frame.setVisible(true);
    }
}
```

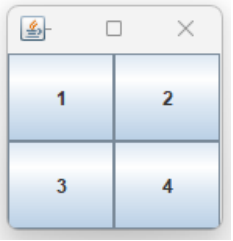

image 1

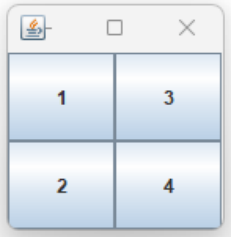

image 2

Question 5 [8]

Consider the following ***Player*** class:

```
public class Player {
    int jersey;
    String name, type;
    public Player(int jersey, String name, String type)
    {
        this.jersey = jersey;
        this.name = name;
        this.type = type;
    }
}
```

Now complete **only the missing codes** for the following ***comparator_main*** class:

```
public class comparator_main {
    public static void main(String[] args)
    {
        // Task 1: Create an empty Arraylist of Player type

        /* Task 2: Using the add() method add objects with the followings to the
                   Arraylist
                   55, "Karim", "Bangladesh"
                   14, "Ponting", "Australia"  */

        /* Task 3: Sort the Arraylist in, ascending order of jersey number using a
                   comparator for comparing objects of Player class [You can also
                   define the Comparator as a separate class if you want]
        */
    }
}
```

Or,

Question 5 [6 + 2]

A. Consider the ***MyStack*** class that can keep at most five characters in the stack. The ***Producer*** thread pushes an element to it while the ***Consumer*** thread pops from it. However, the Producer thread cannot push an element while there are already five elements in it, and it must wait then until an element is popped. Similarly, the Consumer must wait to pop any element till the stack is empty. For implementing the above scenario, write codes for the following tasks to the mentioned sections of the following code snippets:

i. Add wait() method in *pop()* in *MyStack* class

ii. Start threads for object *p* and *c* in *StackTest* class

iii. Add sleep() in *run()* methods of *Producer* and *Consumer* classes

N.B. *Just write only the added lines of codes for each class*

```java
class MyStack {
    private int idx = 0;
    private char[] data = new char[6];

    public synchronized void push(char c) {
        this.notify();
        if (idx != 5) {
            data[idx] = c;
            idx++;
        }
    }

    public synchronized char pop() {
        if (idx == 0) {
            // Lines of code for A;
        }
        idx--;
        return data[idx];
    }
}
```

```java
public class StackTest {
    public static void main(String[] args) {
        MyStack s = new MyStack();
        Producer p = new Producer(s);
        Consumer c = new Consumer(s);
        // Lines of code for B;
    }
}
```

```java
class Producer implements Runnable {
    private MyStack stack;

    public Producer(MyStack s) {
        stack = s;
    }

    public void run() {
        char c;
        for (int i = 0; i < 50; i++) {
            c = (char)(Math.random() * 26 + 'A');
            stack.push(c);
            System.out.println("Producer: " + c);
            // Lines of code for C;
        }
    }
}
```

```java
class Consumer implements Runnable {
    private MyStack stack;

    public Consumer(MyStack s) {
        stack = s;
    }

    public void run() {
        char c;
        for (int i = 0; i < 50; i++) {
            c = stack.pop();
            System.out.println("Consumer: " + c);
            // Lines of code for C;
        }
    }
}
```

B. Briefly explain the Thread state diagram.